

Optimizing Access Cost for Top-k Queries over Web Sources: A Unified Cost-based Approach

Seung-won Hwang and Kevin Chen-Chuan Chang
Department of Computer Science
University of Illinois at Urbana-Champaign
{shwang5, kcchang}@uiuc.edu

We study the problem of supporting ranked queries in middleware environments, where queries are evaluated over multiple sources. In particular, we study Web middleware scenarios, querying over various Web sources. To motivate, consider a Web “travel agent” scenario for finding restaurants and hotels, as Examples 1 and 2 illustrate. (We use this real scenario as “benchmark” queries for experiments as well.) In particular, how to access sources with different capabilities and costs, to answer queries efficiently? As our Web middleware coordinates various sources, each source *access* will incur network communication and server computation. This paper aims at optimizing such access costs—which dominate the overall query processing (like I/O in relational DBMS).

Example 1: To find top-5 restaurants (say, in the Chicago area) that are highly-rated and close to “myaddr,” a user may ask a ranked query Q_1 (in SQL-like syntax):

```
select name from restaurant r
order by min(rating(r.stars), close(r.addr, myaddr))
stop after 5 (Query  $Q_1$ )
```

For query answering, our middleware will access Web sources to evaluate the *predicates*, e.g., *rating*, into scores in $[0:1]$, which are then aggregated by some *scoring function* \mathcal{F} , e.g., $\mathcal{F} = \min$, to return the highest-scored 5 restaurants.

Our middleware can use various sources in query answering: Figure 1(a) shows one possible scenario: For evaluating *close*: superpages.com is capable of 1) returning the *close* score for a specific restaurant (“random access”) and 2) returning restaurants in their descending order of scores (“sorted access”). For *rating*: dineme.com similarly provides both sorted and random accesses.

The middleware will coordinate these accesses to find the top results. Such accesses are typically expensive (as compared to local computations) with varying costs: To characterize, Figure 1(a) shows the average access latency (thus including both network and server times) of both

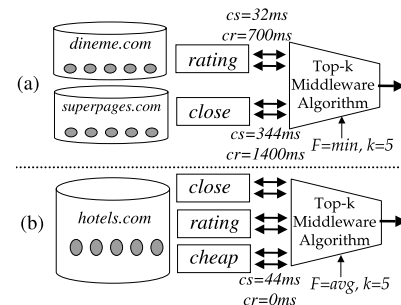


Figure 1. Scenarios for (a) Q_1 and (b) Q_2 .

sorted and random access (denoted cs and cr respectively) for each predicate: In this scenario, random accesses are more expensive in both sources (i.e., $cr > cs$), but with different actual scales (i.e., cr) and ratios (i.e., $\frac{cr}{cs}$). ■

Example 2: Consider query Q_2 for the top-5 hotels that are close, with high star-rating, yet within the budget:

```
select name from hotel h
order by avg(close(h.addr, myaddr),
rating(h.stars), cheap(h.price))
stop after 5 (Query  $Q_2$ )
```

Figure 1(b) describes another scenario, with hotels.com providing sorted access to all predicates. In this setting, since a sorted access (e.g., for *close*) also retrieves all attributes of a hotel (e.g., “stars” and “price”), the subsequent random accesses¹ to the same hotel are essentially of zero access costs—e.g., using “stars” and “price,” the middleware can locally compute *rating* and *cheap*. This scenario significantly contrasts with expensive random accesses of Example 1. ■

Our goal is to develop middleware *algorithms* to minimize access costs. While there have been many middleware algorithms e.g., FA [4, 10], TA, CA, NRA, TA_Z [5], Quick-Combine[6], Stream-Combine [7], SR-Combine

¹In a middleware, random accesses to an object h can only occur after h is first “seen” from sorted accesses— or, “no wild guess” [5].

Sorted Access	Random Access		
	cheap: $cr_i = 1$	expensive: $cr_i = h$	impossible: $cr_i = \infty$
cheap: $cs_i = 1$	FA, TA Quick-Combine	CA, SR-Combine	NRA, Stream-Combine
expensive: $cs_i = h$?	FA, TA SR-Combine	NRA, Stream-Combine
impossible:	T A _z , MPro, Upper	T A _z , MPro, Upper	X

Figure 2. Access scenarios and algorithms.

Framework NC(Q, \mathcal{D}): Necessary Choices

Input: query $Q = (\mathcal{F}(p_1, \dots, p_m), k)$,
database $\mathcal{D} = \{u_1, \dots, u_n\}$

Output: \mathcal{K} , top- k objects from \mathcal{D} w.r.t. to \mathcal{F}

- 1) $\mathcal{P} \leftarrow \phi$; //accesses-so-far
- 2) $\mathcal{K}_{\mathcal{P}} \leftarrow \{v_1, \dots, v_k \mid \text{top-}k \text{ from } \mathcal{D} \text{ ranked by } \overline{\mathcal{F}}_{\mathcal{P}}[\cdot]\}$;
- 3) **while** ($U \leftarrow \{v_j \mid v_j \in \mathcal{K}_{\mathcal{P}}; v_j \text{ is incomplete}\}$)
- 4) $v_j \leftarrow$ any object in U ; //e.g., the highest-ranked
- 5) $N_j \leftarrow \{sa_i, ra_i(v_j) \mid p_i[v_j] \text{ is undetermined by } \mathcal{P}\}$;
alternatives $\leftarrow N_j$;
- 6) *Select* access A from alternatives; //access selection.
- 7) perform A ; update $\mathcal{K}_{\mathcal{P}}$; $\mathcal{P} \leftarrow \mathcal{P} \cup \{A\}$;
- 8) return $\mathcal{K} = \mathcal{K}_{\mathcal{P}}$;

Figure 3. Framework NC.

[1], MPro [3], and Upper [2], supporting different subspace of scenarios as Figure 2 illustrates, we argue they do not satisfy Web querying requirements, by compromising *generality* and *adaptivity*: First, they have mostly been designed with *specific* cost scenarios in mind (e.g., FA for scenarios where sorted access and random access have same “uniform” costs for all predicates), while Web sources are *heterogeneous* with widely varying access capabilities and cost (e.g., as real sources show in Figure 1). Second, they largely lack systematic runtime optimization, while Web cost scenarios dynamically change over time (e.g., depending on source load and availability).

Toward this goal of general yet adaptive framework, we take a “cost-based optimization” approach— By dynamic search over a space Ω of possible algorithms, cost-based optimization is general across virtually all cost scenarios, yet adaptive to the specific one at runtime. While such optimization has been “taken for granted” for relational queries from early on [9], it has been clearly lacking for ranked queries, due to the following challenges: To begin with, what is a “complete” yet “focused” space Ω of top- k algorithms, to search over? Given the space, what is an effective search scheme to identify the optimal algorithm in Ω ?

Our work realizes such cost-based optimization for top- k queries, accomplished by systematically addressing such challenges: First, we develop a focused algorithm space to search over, without compromising generality. To induce this space, our approach hinges on developing an abstract *framework*, inspired by relational algebraic framework with logical operators: Our framework NC (Figure 3) identifies “logical tasks” for top- k queries such that query processing is *equivalent* to fulfilling a set of (necessary and atomic) tasks $\{w_1, \dots, w_n\}$. With this view, Framework NC achieves both *generality* and *specificity* by focusing *only* on unsatisfied tasks— As Figure 3 illustrates, during

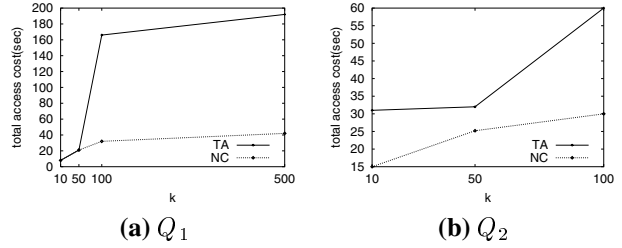


Figure 4. Comparison of total access costs.

processing, when a set of accesses \mathcal{P} has been performed, Framework NC focuses on a set of accesses N_j , *specifically* satisfying unsatisfied w_j . Second, with complete and focused search space Ω defined, we develop systematic optimization schemes, which are empirically validated to well balance both the overhead and the quality of optimization.

In summary, our contributions are as follows:

- **Conceptual unification:** By realizing cost-based optimization, our framework unifies and generalizes beyond existing top- k algorithms, as our extended report [8] discusses.
- **Practical optimization:** By dynamic search over the space, our framework indeed optimizes to the specific access costs at run time— Figure 4 shows how our framework outperforms existing algorithms in our Web benchmark scenarios. However, we stress that, while we study Web querying, our approach is generally applicable in any middleware environments (e.g., multimedia systems [10]), where access costs are significant— Our extended report [8] reports our extensive evaluation over a wider range of synthesized middleware settings as well.

References

- [1] W. Balke, U. Guentzer, and W. Kiessling. On real-time top- k querying for mobile services. In *CoopIS 2002*, 2002.
- [2] N. Bruno, L. Gravano, and A. Marian. Evaluating top- k queries over web-accessible databases. In *ICDE 2002*, 2002.
- [3] K. C.-C. Chang and S. Hwang. Minimal probing: Supporting expensive predicates for top- k queries. In *SIGMOD 2002*, 2002.
- [4] R. Fagin. Combining fuzzy information from multiple systems. In *PODS 1996*, 1996.
- [5] R. Fagin, A. Lote, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS 2001*, 2001.
- [6] U. Guentzer, W. Balke, and W. Kiessling. Optimizing multi-feature queries in image databases. In *VLDB 2000*, 2000.
- [7] U. Guentzer, W. Balke, and W. Kiessling. Towards efficient multi-feature queries in heterogeneous environments. In *ITCC 2001*, 2001.
- [8] S. Hwang and K. C.-C. Chang. Enabling Cost-based Optimization for Top- k Queries: A Unified Framework *Technical Report*, University of Illinois, UIUCDCS-R-2003-2324, 2003.
- [9] P. Selinger, M. Astrahan, D. Chamberlin, R. Lorie, and T. Price. Access Path Selection in a Relational Database Management System. In *SIGMOD 1979*, 1979.
- [10] E. L. Wimmers, L. M. Haas, M. T. Roth, and C. Braendli. Using Fagin’s algorithm for merging ranked results in multimedia middleware. In *CoopIS 1999*, 1999.